R U T C O R
R ESEARCH
R E P O R T

ACCELERATED ALGORITHM FOR PATTERN DETECTION IN LOGICAL ANALYSIS OF DATA

Sorin Alexe[a]
Peter L. Hammer[b]

RRR 59-2001, DECEMBER, 2001

RUTCOR
Rutgers Center for
Operations Research
Rutgers University
640 Bartholomew Road
Piscataway, New Jersey
08854-8003
Telephone: 732-445-3804
Telefax: 732-445-5472
Email: rrr@rutcor.rutgers.edu

[a] *RUTCOR*
Rutgers, the State University of New Jersey
640 Bartholomew Road
Piscataway, NJ 08854-8003
salexe@rutcor.rutgers.edu
[b] *RUTCOR*
Rutgers, the State University of New Jersey
640 Bartholomew Road
Piscataway, NJ 08854-8003
hammer@rutcor.rutgers.edu

RUTCOR RESEARCH REPORT
RRR 59-2001, DECEMBER, 2001

ACCELERATED ALGORITHM FOR PATTERN DETECTION IN LOGICAL ANALYSIS OF DATA

Sorin Alexe Peter L. Hammer

Abstract. Sets of "positive" and "negative" points (observations) in n-dimensional discrete space given along with their non-negative integer multiplicities are analyzed from the perspective of the Logical Analysis of Data (LAD). A set of observations satisfying upper and/or lower bounds imposed on certain components is called a positive pattern if it contains some positive observations and no negative one. The number of variables on which such restrictions are imposed is called the degree of the pattern. A total polynomial algorithm is proposed for the enumeration of all patterns of limited degree, and special efficient variants of it for the enumeration of all patterns with certain "sign" and "coverage" requirements are presented and evaluated on a publicly available collection of benchmark datasets.

Acknowledgements: The partial support of the Office of Naval Research (Grant N00014-92-J-1375) and the Center for Discrete Mathematics and Computer Science are gratefully acknowledged.

1 Introduction

In a variety of problems occurring in data mining, machine learning, and other areas, a finite set Ω of points is given in a discrete space, and it is required to enumerate the cardinalities of the subsets of points of Ω included in the family of all intervals of the given space which satisfy certain conditions. The object of this paper is to provide an efficient algorithm for the solution of this type of problems.

In section 2 we provide the basic definitions of intervals in a discrete space and some of their characteristic properties. In section 3 we assume that a finite set of points Ω is given in a discrete space L, and provide an efficient algorithm for enumerating all the intervals of $\boldsymbol{L}$ along with the cardinalities of their intersections with Ω. Section 4 is devoted to the complexity analysis of the procedure, and presents the results of computational experiments carried out on several publicly available machine learning datasets. Section 5 is dedicated to two applications. The first one concerns the detection of dense regions of points in datasets -- a problem frequently occurring in image recognition, tomography, and other areas. The second application concerns a central problem in the "Logical Analysis of Data": the enumeration of positive and negative patterns (intervals satisfying some typical requirements). We report results of computational experiments showing the high efficiency of the proposed method for these applications.

2 Intervals and their characteristics

For any positive integer k let $[k]$ denote the ordered set $\{1,2,...,n\}$, and let us define the *discrete space* L as $L=[k_1]\times[k_2]\times...\times[k_n]$. We shall consider below sets of points $\Omega=\{X^1,X^2,...,X^t\}\subseteq L$, $X^j=(x_1^j,x_2^j,...,x_n^j)$, along with sets $\{m^1,m^2,...,m^t\}$ of positive integers, called *multipliers*, associated to the points in Ω.

For any $U=(u_1,u_2,...,u_n)$ and $V=(v_1,v_2,...,v_n)\in L$ we shall say that $U\le V$ if $u_j\le v_j$ for all $j\in\{1,2,...,n\}$. If $U\le V$ we shall say that

$$I[U,V]=\{x\in L\,|\,U\le x\le V\} \tag{1}$$

is the interval *defined by U* and *V*.

For any $U=(u_1,u_2,...,u_n)$ and $V=(v_1,v_2,...,v_n)\in L$ let us denote by $\min(U,V)$ the element of L defined by:

$$\min(U,V)=(\min(u_1,v_1),\min(u_2,v_2),...,\min(u_n,v_n)) \tag{2}$$

The element $\max(U,V)$ of L is defined in a similar way:

$$\max(U,V)=(\max(u_1,v_1),\max(u_2,v_2),...,\max(u_n,v_n)) \tag{3}$$

For any $U,V \in L$ we shall define the *hull* $H[U,V]$ of the set $\{U,V\}$ as the interval defined by $\min(U,V)$ and $\max(U,V)$

$$H[U,V] = I[\min(U,V), \max(U,V)] \tag{4}$$

Clearly, $H[U,V] = I[U,V]$ if and only if $U \leq V$.

If I is an interval of L defined by $U = (u_1, u_2, ..., u_n)$ and $V = (v_1, v_2, ..., v_n)$, (with $U \leq V$) then its *degree* is

$$\deg(I) = card(\{i \in \{1,2,...,n\} \mid v_i - u_i < k_i\}) \tag{5}$$

The *prevalence* of an interval *I* is the sum of the multiplicities of the points in $I \cap \Omega$:

$$p(I) = \sum_{X^j \in I} m^j \tag{6}$$

We define the *n*-dimensional *distribution matrix*

$$M = (m_X)_{X \in L} \tag{7}$$

whose elements m_X are equal to the multiplicities m^j if $X = X^j \in \Omega$, and to 0 if $X \notin \Omega$.

Proposition 1 *Let* $I = I[U,V]$ *be an interval of L. The prevalence of the interval I is:*

$$p(I) = \sum_{U \leq X \leq V} m_X \tag{8}$$

Proof. Indeed,

$$p(I) = \sum_{X^j \in \Omega} m^j = \sum_{X \in I \cap \Omega} m_X = \sum_{X \in I} m_X = \sum_{U \leq X \leq V} m_X .$$

For the example shown in figure 1, if $n = 2, k_1 = 3, k_2 = 4$ and if Ω consists of the points $X^1 = (0,4), X^2 = (1,2), X^3 = (1,3), X^4 = (2,2), X^5 = (2,3), X^6 = (3,0)$, all multiplicities being equal to 1, let us consider the intervals I_1 defined by $1 \leq x_1 \leq 2$, I_2 defined by $2 \leq x_2 \leq 3$ and I_3 defined by $1 \leq x_1 \leq 2$ and $2 \leq x_2 \leq 3$. It is easy to notice that $p(I_1) = p(I_2) = p(I_3) = 4$.

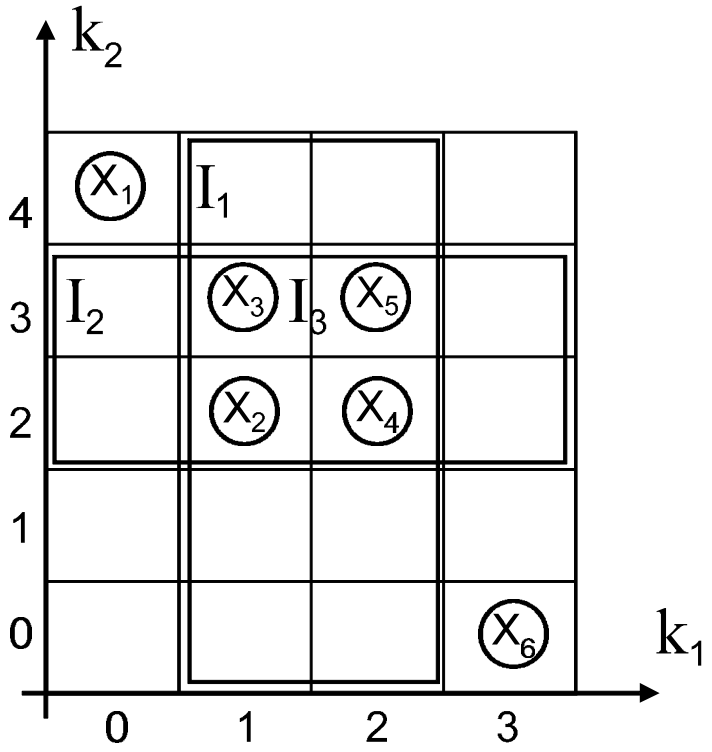

Figure 1

The distribution matrix *M* corresponding to this dataset is

$$M = \begin{bmatrix} m_{(0.4)} & m_{(1.4)} & m_{(2.4)} & m_{(3.4)} \\ m_{(0.3)} & m_{(1.3)} & m_{(2.3)} & m_{(3.3)} \\ m_{(0.2)} & m_{(1.2)} & m_{(2.2)} & m_{(3.2)} \\ m_{(0.1)} & m_{(1.1)} & m_{(2.1)} & m_{(3.1)} \\ m_{(0.0)} & m_{(1.0)} & m_{(2.0)} & m_{(3.0)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

In the following section we shall make extensive use of the *n*-dimensional matrix Π^V with

$$p_U^V = \sum_{X \in H[U,V]} m_X \ . \tag{10}$$

In our example the matrix Π^{V^0} corresponding to the maximal element $V^0=(3,4)\in L$ is

$$\Pi^{V^0}=\begin{bmatrix} p_{(0,4)}^{(3,4)} & p_{(1,4)}^{(3,4)} & p_{(2,4)}^{(3,4)} & p_{(3,4)}^{(3,4)} \\ p_{(0,3)}^{(3,4)} & p_{(1,3)}^{(3,4)} & p_{(2,3)}^{(3,4)} & p_{(3,3)}^{(3,4)} \\ p_{(0,2)}^{(3,4)} & p_{(1,2)}^{(3,4)} & p_{(2,2)}^{(3,4)} & p_{(3,2)}^{(3,4)} \\ p_{(0,1)}^{(3,4)} & p_{(1,1)}^{(3,4)} & p_{(2,1)}^{(3,4)} & p_{(3,1)}^{(3,4)} \\ p_{(0,0)}^{(3,4)} & p_{(1,0)}^{(3,4)} & p_{(2,0)}^{(3,4)} & p_{(3,0)}^{(3,4)} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 5 & 4 & 2 & 0 \\ 5 & 4 & 2 & 0 \\ 6 & 5 & 2 & 1 \end{bmatrix}. \tag{11}$$

For example, as $L=[3]\times[4]$ and $(1,1)\geq(3,4)$, we have $p(I[(1,1),(3,4)])=p_{(1,1)}^{(3,4)}=4$.

3 Prevalence calculation

One of the key questions in a variety of applications centering around data analysis problems consists in finding all those intervals which satisfy certain conditions. The requirement that the prevalence should be sufficiently large is normally included among the conditions describing those intervals which are of interest. We shall return to this topic in the application section 5 of this paper.

The aim of this section is to describe an algorithm for enumerating all the intervals $I[U,V]$, with $U\leq V\in L$, along with their prevalences.

As a matter of fact, for every fixed $V^*\in L$, we shall determine the prevalences of all the hulls $H[U,V^*]$, whether the relation $U\leq V^*$ does or does not hold. Clearly, in view of the fact that $H[U,V^*]=I[\min(U,V^*),\max(U,V^*)]$, all the intervals $I[U,V^*],U\leq V^*$ will be present among these hulls $H[U,V^*],U,V^*\in L$. The computational advantage of carrying out these apparently unnecessary additional calculations will become apparent from the description of the algorithm.

For each V^*, we shall represent the calculated prevalences of the hulls $H[U,V^*]$, in the form of a *prevalence matrix*.

The discussion will be split in three parts. In the first part we shall present a simple way for calculating the initial prevalence matrix corresponding to the maximal element $V^0=(k_1,k_2,...,k_n)\in L$ having as elements the prevalences of $H[U,V^0]$, for every U in L. In the second part we shall present a "Gray code" which lists all the elements in L, in the form of a sequence $V^0,V^1,...,V^q$, where for each $i\in\{0,1,...,q-1\}$, the element V^{i+1} differs from V^i by a unit vector. In the third part, using the Gray code enumeration of the elements V^i, a simple transformation is presented for calculating directly the prevalences of all the hulls $H[U,V^{i+1}]$ from those of the hulls $H[U,V^i]$. Since the prevalence matrix Π^{V^i} obtained in this way is

defined in terms of the vector V^i, the vector V^i will be called its *basis* and the Gray code used in the second part of the algorithm can be viewed as a basis transformation method.

3.1 Initial prevalence matrix

As initial basis we shall take the maximal element of L, $V^0 = (k_1, k_2, ..., k_n)$. In order to calculate the matrix Π^{V^0} we shall start with the matrix M defined in (7), and denote it for the sake of recurrence as $M^0 = \left(\left(m_X^0\right)\right)_{X \in L}$. Let us associate to the matrix M^0 $M^1 = \left(\left(m_X^1\right)\right)_{X \in L}$ with elements defined by:

$$m^1_{(x_1, x_2, \ldots x_n)} = \sum_{j=x_1}^{k_1} m^0_{(j, x_2, \ldots x_n)} \tag{12}$$

Similarly, we shall define the matrices $M^2, M^3, ..., M^n$ recursively by:

$$m^i_{(x_1, x_2, ..., x_i, ..., x_n)} = \sum_{j=x_1}^{k_1} m^0_{(j, x_2, \ldots x_n)} \,. \tag{13}$$

The following simplified formulas follow directly from (13):

- If $x_i = k_i$,

$$m^i_{(x_1, x_2, ..., x_i, ..., x_n)} = m^{i-1}_{(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)} \tag{14}$$

- If $x_i < k_i$,

$$m^i_{(x_1, x_2, ..., x_i, ..., x_n)} = m^i_{(x_1, x_2, ..., x_{i-1}, x_i + 1, x_{i+1}, ..., x_n)} + m^{i-1}_{(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)} \tag{15}$$

Indeed, if $x_i < k_i$,

$$\begin{aligned} m^i_{(x_1, x_2, ..., x_i, ..., x_n)} &= \sum_{j=x_i}^{k_i} m^{i-1}_{(x_1, x_2, ..., x_{i-1}, j, x_{i+1} ..., x_n)} = \\ &= \sum_{j=x_i+1}^{k_i} m^{i-1}_{(x_1, x_2, ..., x_{i-1}, j, x_{i+1} ..., x_n)} + m^{i-1}_{(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1} ..., x_n)} = \\ &= m^i_{(x_1, x_2, ..., x_{i-1}, x_i + 1, x_{i+1} ..., x_n)} + m^{i-1}_{(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1} ..., x_n)} \,. \end{aligned}$$

It is easy to check now that

Proposition 2 *The matrix* Π^{V^0} *is equal to* M^n*, and is constructed by the algorithm described above, using at most* $n \prod_{i=1}^{n} (k_i + 1)$ *additions.*

Proof. By using repeatedly formula (13), we have:

$$m^n_{(u_1, u_2, ..., u_n)} = \sum_{x_n = u_n}^{k_n} m^{n-1}_{(u_1, u_2, ..., u_{n-1}, x_n)} = \ldots = \sum_{x_1 = u_1}^{k_1} \ldots \sum_{x_n = u_n}^{k_n} m^{n-1}_{(x_1, x_2, ..., x_{n-1}, x_n)} = \sum_{U \le X \le V^0} m_X = p_U^{V^0} \,.$$

Clearly, if M^{i-1} is known, in order to compute the matrix M^i by using (14) and (15) we have to perform $(k_1+1)\cdot(k_2+1)\cdot\ldots\cdot(k_{i-1}+1)\cdot k_i\cdot(k_{i+1}+1)\cdot\ldots\cdot(k_n+1)\le\prod_{i=1}^{n}(k_i+1)$ additions. As the process has to be repeated for each of the *n* coordinates, the total number of additions needed to complete the construction of the matrix Π^{V^0} is at most $n\cdot\prod_{i=1}^{n}(k_i+1)$.

In the example above

$$M^0=M=\begin{bmatrix}1&0&0&0\\0&1&1&0\\0&1&1&0\\0&0&0&0\\0&0&0&1\end{bmatrix},\quad M^1=\begin{bmatrix}1&0&0&0\\2&2&1&0\\2&2&1&0\\0&0&0&0\\1&1&1&1\end{bmatrix},\quad M^2=\Pi^{V^0}=\begin{bmatrix}1&0&0&0\\3&2&1&0\\5&4&2&0\\5&4&2&0\\6&4&3&1\end{bmatrix}. \quad (16)$$

A pseudo-code for calculating Π^{V^0} is presented below.

Algorithm 1 for constructing Π^{V^0} starting with *M*

H:=*M*;

for *i* from 1 to *n* do

for all $X=(x_1,x_2,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)$ do

for *j* from k_i-1 downto 0 do

$$h_{(x_1,x_2,\ldots,j,\ldots x_n)}:=h_{(x_1,x_2,\ldots,j,\ldots x_n)}+h_{(x_1,x_2,\ldots,j+1,\ldots x_n)};$$

$\Pi^{V^0}:=H$.

3.2 A Gray code basis transformation

Gray codes (see [7]), i.e.. sequences of binary vectors $V^1,V^2,\ldots,V^q$ such that V^i and V^{i+1} differ in a single component, have been extensively studied ([1], [4], [6], [9], [12]). We shall discuss in this section an extension of the Koda and Ruskey ([11]) Gray code enumeration method for binary sequences.

The proposed extension of Gray code preserves the property that it enumerates the elements of L in such a way that two consecutive basis are always at "block distance" 1; the *block distance* of two points $U=(u_1,u_2,...,u_n)$ and $V=(v_1,v_2,...,v_n)$ in L is defined as:

$$d(U,V)=\sum_{i=1}^{n}|u_i-v_i| \tag{17}$$

It is easy to notice that d satisfies the usual distance axioms.

In the proposed enumeration method we associate to each basis $V=(v_1,v_2,...,v_n)\in L$, "transition" vector $T^V=(t_1^V,t_2^V,...,t_n^V)$, with $t_i^V\in\{-1,+1\}$.

As initial V^0 we shall take $V^0=(k_1,k_2,...,k_n)\in L$, and as initial T^0 we shall take $T^0=(-1,-1,...,-1)$.

Let us define the set

$$S^r=\{i\in\{1,2,...,n\}\mid 0\le v_i+t_i\le k_i\} \tag{18}$$

If $S^r=\varnothing$ the process is finished.

Otherwise, let

$$i^*=\max_{i\in S^r} i\,, \tag{19}$$

and let us define the element $V^{r+1}=(v_1^{r+1},v_2^{r+1},...,v_n^{r+1})$ and the corresponding $T^{r+1}=(t_1^{r+1},t_2^{r+1},...,t_n^{r+1})$ by putting

$$v_i^{r+1}=\begin{cases} v_i^r, & \text{if } i\ne i^* \\ v_{i^*}^r+t_{i^*}^r, & \text{if } i=i^* \end{cases}, \tag{20}$$

and

$$t_i^{r+1}=\begin{cases} t_i^r, & \text{if } i\ne i^* \\ -t_i^r, & \text{if } i=i^* \end{cases}. \tag{21}$$

It has been seen in [11] that the above-described enumerative process satisfies the following conditions:

(i) every element of L appears as a basis;
(ii) no element of L appears twice as basis;
(iii) the block distance of two consecutive bases is equal to 1.

Algorithm 2 for the generation of the extended Gray code

Step 1.

$V=(v_1,v_2,...,v_n):=(k_1,k_2,...,k_n)$;

$T=(t_1,t_2,...,t_n):=(-1,-1,...,-1)$;

Step 2.

Repeat (forever)

{ $S:=\{i\in\{1,2,...,n\}\mid 0\le v_i+t_i\le k_i\}$;

if $S^r=\varnothing$ then STOP;

$i^*=\max S$;

$v_{i^*}:=v_{i^*}+t_{i^*}$;

for all $i>i^*$ do

$t_i:=-t_i$;

Print(V);

};

3.3 Iterative calculation of prevalence matrices

In this section we shall describe a method for the calculation of all the *n*-dimensional matrices $\Pi^V, V\in L$, and shall obtain in this way all the interval prevalences. The proposed method involves a series of iterative steps, each step of the iteration being defined by its basis $V=(v_1,v_2,...,v_n)\in L$. At the iteration corresponding to a particular basis $V\in L$ we shall calculate an *n*-dimensional matrix Π^V having $(k_1+1)\cdot(k_2+1)\cdot...\cdot(k_n+1)$ elements p_U^V, and shall show that

$$p\left(H[U,V]\right)=p_U^V. \tag{22}$$

These calculations will be facilitated by the fact that V is at lock distance 1 from the basis V', proceeding it in the Gray code, and that $\Pi^{V'}$ was already calculated at the previous step.

Using the transformation formulas (20) and (21) we can see that if i^* is the index corresponding to V^r by (19) and if $t_{i^*}=+1$ then

$$p_U^{V^{r+1}}=\begin{cases} p^{V^r}_{u_1,u_2,...,u_{i^*},...,u_n}-p^{V^r}_{u_1,u_2,...,v^r_{i^*},...,u_n}, & \text{if } u_{i^*}>v^r_{i^*} \\ p^{V^r}_{u_1,u_2,...,u_{i^*},...,u_n}+p^{V^{r+1}}_{u_1,u_2,...,v^r_{i^*}+1,...,u_n}, & \text{if } u_{i^*}\le v^r_{i^*} \end{cases}. \tag{23}$$

Similarly, if $t_{i^*}=-1$ then

$$p_U^{V^{r+1}}=\begin{cases} p^{V^r}_{u_1,u_2,...,u_{i^*},...,u_n}-p^{V^r}_{u_1,u_2,...,v^r_{i^*},...,u_n}, & \text{if } u_{i^*}<v^r_{i^*} \\ p^{V^r}_{u_1,u_2,...,u_{i^*},...,u_n}+p^{V^{r+1}}_{u_1,u_2,...,v^r_{i^*}-1,...,u_n}, & \text{if } u_{i^*}\ge v^r_{i^*} \end{cases}. \tag{24}$$

Clearly, the computation of $\Pi^{V^{r+1}}$ can be done by executing $\prod_{i=1}^{n}(k_i+1)$ additions. It can be shown that

Proposition 3 Let V^0 be the maximal element of L, Π^{V^0} the matrix defined by algorithm 1, let $V^0, V^1, ..., V^q$ be the Gray code defined by algorithm 2 and let $\Pi^{V^0}, \Pi^{V^1}, ..., \Pi^{V^q}$ be the matrices defined by formulas (23), (24). Then for any $U \in L$, with $U \le V^r$ $(r = 1,2,...,q)$

$$p\left(H\left[U, V^r\right]\right) = p_U^{V^r}. \tag{25}$$

Proof. Clearly, the matrix Π^{V^0} satisfies relation (25). Let us assume now that relation (25) is valid at step *r* of the iterative process. Four cases can be distinguished according to the value of t_{i^*} and the relation between u_{i^*} and v_{i^*}. Since the four cases have similar proofs we shall present only one of them. Considering for example $t_{i^*} = +1$ and $u_i > v_{i^*}$ we have

$$\begin{aligned}
&p_U^{V^{r+1}} = p_{(u_1,u_2,...,u_{i^*},...,u_n)}^{V^r} - p_{(u_1,u_2,...,v_{i^*}^r,...,u_n)}^{V^r} = \\
&= p\left(H[(u_1,u_2,...,u_{i^*},...,u_n), V^r]\right) - p\left(H[(u_1,u_2,...,v_{i^*}^r,...,u_n), V^r]\right) = \\
&= p\left(H\left[\left[(u_1,u_2,...,u_{i^*},...,u_n), (v_1^r, v_2^r, ..., v_{i^*}^r + 1, ..., v_n^r)\right]\right)\right) = \\
&= p\left(H\left(\left[\left(u_1,u_2,...,u_{i^*},...,u_n\right), \left(v_1^{r+1}, v_2^{r+1}, ..., v_{i^*}^{r+1}, ..., v_n^{r+1}\right)\right]\right)\right) = \\
&= p\left(H\left(\left[U, V^{r+1}\right]\right)\right).
\end{aligned}$$

Similar proofs hold for each other cases.

In our example, using the same row and column indexing as in (11), starting with $V^0 = (3,4)$ and the matrix Π^{V^0} calculated in (11), we define V^1 to be $V^1 = (3,3)$. We calculate now Π^{V^1} by subtracting the row with index 4 from the rows 0, 1, 2, 3 and add the new row 3 to the rows with index 4, obtaining

$$\Pi^{V^1} = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 4 & 4 & 2 & 0 \\ 4 & 4 & 2 & 0 \\ 5 & 5 & 3 & 1 \end{bmatrix}.$$

Note that, when transforming a matrix Π^{V^r} to $\Pi^{V^{r+1}}$, every element of the matrix is changed, each of these changes involving a single operation of addition or subtraction.

Algorithm 3 Enumeration of all matrices $\Pi^V, V \in L$.

// If $d(V,V')=1$ then Π^V and $\Pi^{V'}$ are computed consecutively.

Step 1. $V := V^0$;

Compute Π^V using algorithm 1.

Step 2.

Repeat

{ If V is the last basis (i.e. $V=(0,0,...,0)$) then STOP.

Let V' be the basis which follows V and the corresponding i^* and T

for all $X=(x_1,x_2,...,x_{i^*-1},0,x_{i^*+1},...,x_n)$ do

{ if $t_{i^*}=+1$ then

{ for all $j > v_i$ do

$$p^{V'}_{(x_1,x_2,...,j,...x_n)} = p^{V}_{(x_1,x_2,...,j,...x_n)} - p^{V}_{(x_1,x_2,...,v_i,...x_n)};$$

for all $j \le v_i$ do

$$p^{V'}_{(x_1,x_2,...,j,...x_n)} = p^{V}_{(x_1,x_2,...,j,...x_n)} + p^{V'}_{(x_1,x_2,...,v_i+1,...x_n)};$$

};

if $t_{i^*}=-1$ then

{ for all $j < v_i$ do

$$p^{V'}_{(x_1,x_2,...,j,...x_n)} = p^{V}_{(x_1,x_2,...,j,...x_n)} - p^{V}_{(x_1,x_2,...,v_i,...x_n)};$$

for all $j \ge v_i$ do

$$p^{V'}_{(x_1,x_2,...,j,...x_n)} = p^{V}_{(x_1,x_2,...,j,...x_n)} + p^{V}_{(x_1,x_2,...,v_i+1,...x_n)};$$

};

};

Process $(\Pi^{V'})$;

};

4 Complexity analysis and computational results

Since the variables x_i can take (k_i+1) values, on each of the coordinates there are $\frac{(k_i+1)(k_i+2)}{2}$ interval projections. Therefore, the number of intervals of L is

$$N = \frac{1}{2^n}\prod_{i=1}^{n}(k_i+1)(k_i+2). \tag{26}$$

If the n-dimensional matrix was computed, and $V' \in L$ differs from V on the i^{th} coordinate, then the number of additions needed for the computation of the matrix $\Pi^{V'}$ is $\prod_{i=1}^{n}(k_i+1)$. The procedure for computing all the n-dimensional matrices $\Pi^V, V \in L$, starts with

the calculation of Π^{V^0}. Afterwards, the procedure repeats $\left(\prod_{i=1}^{n}(k_i+1)-1\right)$ times the calculation of $\Pi^{V'}$ from the knowledge of its "neighbor" Π^{V}. It follows that the number of additions, denoted by *Op* that are needed to complete the algorithm for calculating the prevalences of all the intervals of L is

$$Op = n\prod_{i=1}^{n}(k_i+1)+\left(\prod_{i=1}^{n}(k_i+1)-1\right)\prod_{i=1}^{n}(k_i+1)=\prod_{i=1}^{n}(k_i+1)\left[N-1+\prod_{i=1}^{n}(k_i+1)\right]\le\prod_{i=1}^{n}(k_i+1)(k_i+2).$$

Consequently,

$$Op \le 2^n N, \tag{27}$$

showing that, for problems having a small number of attributes (which is the case of most real life problems), the algorithm for calculating the prevalences of all intervals is linear in the size of the output.

In order to evaluate the efficiency of the proposed algorithm we have applied the algorithms 1 and 2 for the generation of all intervals of datasets available in the Machine Learning Repository of the University of California at Irvine [1].

Out of the 16 datasets described in ([8]), we have chosen those 5 which have binary outcomes (since these examples will be used also in the computational experiments to be reported in the next section). These 5 datasets are called in the original description "Wisconsin breast cancer", "BUPA liver disorder", "StatLog heart disease", "PIMA Indian diabetes", and "Congressional voting records", and are referred to in this paper as **bcw**, **bld**, **hea**, **pid** and **vot**.

The values of the numerical attributes for all the above data were discretized by partitioning the range of values of each of these variables into 11 consecutive segments, each containing (projections of) approximately equal numbers of observation points. In this way the discretized attribute values of all non-binary variables become 0,1,...,10, and in the definition of the discrete space L each of the k_i's is equal to 1 for the binary variables, and to 10 for the numerical ones.

All computational experiments were carried out using a 1MHz Pentium III processor. The results of the computational experiments are summarized in the following Table 1:

[1] http://www1.ics.uci.edu/~mlearn/MLRepository.html

Dataset	Number of Attributes		Number of Observations	degree	Number of intervals	Total Time (sec)	Time per interval
	Binary	Numerical					
bcw	0	9	683	2	10,349,856	**0**	5.797E-09
				3	24,149,664	**3**	1.217E-07
				4	36,224,496	**163**	4.494E-06
vot	16	0	435	2	1,080	**0**	0
				3	15,120	**0**	0
				4	147,420	**0**	3.392E-07
bld	0	6	345	2	65,340	**0**	0
				3	5,749,920	**2**	3.044E-07
				4	284,621,040	**111**	3.883E-07
hea	3	10	297	2	201,987	**0**	2.97E-07
				3	36,281,547	**2**	4.245E-08
				4	4,300,500,600	**117**	2.71E-08
pid	0	7	455	2	91,476	**0**	0
				3	10,062,360	**3**	2.862E-07
				4	664,115,760	**233**	3.514E-07

Table 1. Time for producing all intervals for degree up to 4

The total time reported in this table includes the enumeration of all the intervals of degree up to 4, along with their prevalences; we have omitted the intervals of degree 1 since obviously they are not too many and their complete enumeration can be done in less then 0.01 seconds.

It can be seen that in spite of the fact that the number of these intervals can be very large (possibly exceeding 4.3 billions), their complete enumeration never exceeds 4 minutes. It can also be seen that the average time for producing all intervals of degree 2 is of the order of a few hundreds of a second, the enumeration of all intervals of degree 3 is on the average less than 2 seconds, and the enumeration of all intervals of degree 4 requires on the average about 2 minutes.

5 Applications

We present below two applications of the proposed interval enumeration procedure in data minning: finding "clumps" of datasets, (i.e. dense regions of observations), and pattern detection for the "Logical Analysis of Data".

5.1 Clumps of datasets

Given a dataset $\Omega \subseteq L$ we shall say that a subset $C \subseteq L$ is a *clump* if it is a maximal union of intervals of L containing a high concentration of points of Ω. Typical applications appear in image recognition, tomography, geological prospecting, etc. In all these situations, the dataset Ω includes a number of points described by their coordinates, and by the presence or absence of an object, a tumor, a mineral, etc. in these points. The clumps of L defined by Ω can be used for approximating the shape of the unknown 2 or 3 - dimensional object.

In order to give a more precise meaning to the concept of clumps, we shall define the *volume* of an interval $I = I[U,V]$ of L as

$$V(I) = \prod_{j \in J} (v_j - u_j + 1) \prod_{j \in J} (k_i + 1), \tag{28}$$

and the *density* of the interval I in Ω as

$$\Delta_\Omega(I) = \frac{p(I)}{V(I.)} \tag{29}$$

In the example shown in Figure 1 the volumes of the intervals I_1, I_2 and I_3 are $V(I_1) = (2-1+1)(4+1) = 10, \quad V(I_2) = (3-2+1)(3+1) = 8, \quad V(I_3) = (2-1+1)(3-1+1) = 4,$ and therefore, their densities are respectively $\Delta_\Omega(I_1) = \frac{2}{5}, \Delta_\Omega(I_2) = \frac{1}{2}, \Delta_\Omega(I_3) = 1.$

In a more formal way, we shall define the d *-clump* C of Ω in L as being the union of all intervals of L whose density in Ω is not smaller then a given d :

$$C = \bigcup_{\Delta_\Omega(I_j) \geq d} I_j. \tag{30}$$

As an example, let us consider the image of a tree root (Fig. 2a). The image of this tree root on a 1159×1402 grid uses 561,435 pixels. Let us consider now a random generated sample consisting of 10% of the pixels (i.e. 56,170 pixels) used by the image of the tree root (Fig. 2b). Let us place now the image of the tree root on a 100×100 grid representing the discrete space L as shown in Fig. 2c. The set Ω will consist of those elements of L which contain black pixels, and the multiplicity of an element will be simply the number of black pixels contained in it. We shall enumerate now all the intervals whose volume is at most 20, and whose density cannot be below some value d . Giving d the values 14, 16, 18, 20 we obtain the d - clumps shown in figure 3 a, b, c, d. It can be seen that the image corresponding to d =18 provides the best reconstruction of the original picture.

Figure 2a

Figure 2b

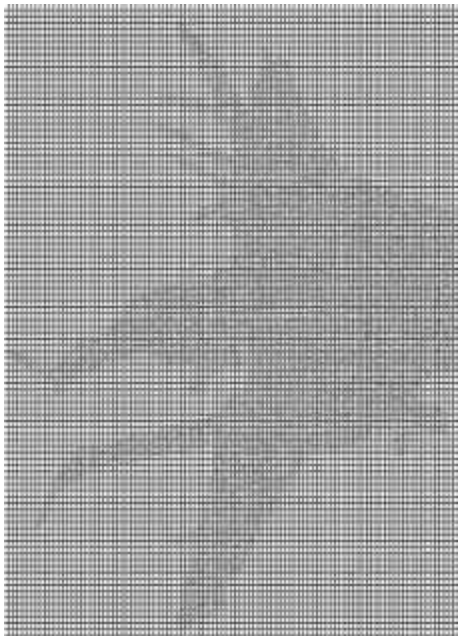

Figure 2c

Figure 3 a (δ=14)

Figure 3 b (δ=16)

Figure 3 c (δ=18)

Figure 3 d (δ=20)

5.2 Logical Analysis of Data

The Logical Analysis of Data (LAD) is a methodology for analyzing collections of observations, each one having a "positive" or a "negative" character. In LAD a dataset Ω is given, consisting of two disjoint sets, Ω^+ and Ω^-, of *positive*, respectively *negative*, points in $\mathbf{R}^n$. The basic question of LAD is to "learn" as much as possible about the structure of the sets Ω^+ and Ω^-. In

particular, LAD can be used for "classification", i.e. for establishing the positive or negative nature of points not included in Ω.

The original version of LAD ([10], [5]) dealt with the case of binary (0-1) data. In a subsequent development ([2], [3]) the methodology was extended to the case of numerical data, based on a transformation which converted each real-valued variable into several binary ones. This "binarization" process transforms a problem with numerical data to a problem having a larger number of binary variables.

The basic tool of LAD on which all its models are built, is the concept of *pattern*, or *interval*. A key algorithmic problem in LAD is the detection of all the intervals which satisfy certain requirements.

In this paper we shall apply the interval enumeration method proposed in the previous sections to the specific requirements of the LAD methodology, thus eliminating the binarization step.

There is an important difference between the intervals considered in the previous sections and the patterns considered in LAD. While the intervals considered in the previous sections did not differentiate between the various points of the dataset, in the case of LAD it is important to restrict the attention to those intervals which contain only (or "almost" only) positive, or only (or "almost" only) negative points.

Let us first introduce some definitions.

An interval I is called a *positive pattern* if $I \cap \Omega^+ \neq \emptyset$ and $I \cap \Omega^- = \emptyset$. Similarly, an interval I is called a *negative pattern* if $I \cap \Omega^- \neq \emptyset$ and $I \cap \Omega^+ = \emptyset$. In the original LAD studies the only patterns of interest were the positive and the negative ones. It turned out latter that in some important classes of applications "almost" positive or "almost" negative patterns play a central role. We shall say that an interval I, with $I \cap \Omega \neq \emptyset$ is *almost positive* or *almost negative* if it has a *sign* sufficiently close to 1, or sufficiently close to 0; here the *sign* $s(I)$ is a number in [0,1] defined by

$$s(I) = \frac{p_{\Omega^+}(I)}{p_{\Omega^+}(I) + p_{\Omega^-}(I)}. \tag{31}$$

Clearly, the positive patterns are simply those intervals whose sign is exactly 1, while the negative patterns are those whose sign is exactly 0.

Since the number of patterns is extremely large, and since not all the patterns have the same significance, LAD requires the detection of only those patterns which satisfy certain requirements. The most common requirements impose limitations on

1. the positive and negative prevalence,
2. the sign of the patterns, and
3. their degree.

A pattern satisfying the above 3 conditions for some specific limitations of prevalence, sign and degree will be called a *feasible pattern* for the given specifications. The limitations vary from problem to problem. Usually, it is required that the positive and negative prevalences (as percentages of the number of elements of Ω^+ and Ω^-) should be sufficiently high, that the sign of the patterns should be sufficiently close to 1 or to 0, and that the degree of the patterns should

be sufficiently low. The low degree requirement is motivated by the need to communicate in a comprehensible way the meaning of patterns to field experts. Additionally, from a computational point of view, the enumeration of low degree patterns is much easier than that of higher degree ones.

In order to apply the interval enumeration method described in this paper to LAD problems, we shall proceed as follows.

If our objective is to enumerate all patterns of degree at most d, and if L is the n-dimensional discrete space, we shall enumerate all $\binom{n}{d}$ subsets of the variables. We shall consider then the projections of Ω on each of these subsets, and for each of these projections we shall enumerate all the feasible intervals.

We shall consider again the same publicly available datasets which were discussed in the previous section. In order to illustrate the computational power of the pattern enumeration method, we shall apply it to the enumeration of several types of patterns present in these datasets.

The three characteristic parameters for the two families consisting respectively of all the positive and all the negative intervals to be constructed, were defined in the following way:

(i) In view of the varying characteristics of the 5 datasets, 2 different sign requirements were imposed, for the almost positive, and almost negative intervals in the 5 datasets. In the case of the **bcw** and **vot** datasets, only positive and negative patterns (i.e. only patterns whose sign is exactly 1 or 0) were considered. For the other datasets, only those almost positive and almost negative patterns which satisfy respectively the sign requirements $s \geq 0.85$ and $s \leq 0.15$, were considered.

(ii) The prevalences of the almost positive patterns considered had to equal at least 10% of the number of positive observations. Similarly, the prevalences of the almost negative patterns considered had to equal at least 10% of the number of negative observations.

(iii) The degrees of the patterns were limited to at most 4.

We have also imposed the requirement that in the final list only those patterns which satisfy a *maximality* condition should be retained. In view of the fact that the set of all feasible patterns is not hereditary, a convenient definition of maximality can be formulated in the following way: A feasible pattern I is called *maximal* if for every feasible pattern J which properly contains it, there exists an interval K which is not feasible and is such that $I \subset K \subset J$.

In all the experiments below we have randomly extracted 50% of the observations to form a "training set", and the patterns were enumerated using only this part of the dataset. The performance of the individual patterns as well as of the entire collection of patterns were tested on the "validation set", consisting of the remaining 50% of the observations.

The results of the computational experiments are summarized in the following Table 2, and lead to some interesting conclusions.

Dataset	Number of Attributes		Number of Observations		Number of Patterns produced			Total Time (sec)	Time per maximal pattern
	Binary	Numerical	Positive	Negative	degree	positive	negative		
bcw	0	9	342	341	2	104	47	0*	0.000
					3	506	418	35	0.038
					4	1205	1660	3646	1.273
vot	16	0	218	217	2	15	0	0*	0.000
					3	137	49	0*	0.000
					4	252	17	1	0.004
bld	0	6	173	172	2	26	2	0*	0.000
					3	106	33	35	0.252
					4	235	192	3337	7.815
hea	3	10	149	148	2	97	109	0*	0.000
					3	726	857	229	0.145
					4	4172	4208	2418	0.289
pid	0	7	228	227	2	41	48	0*	0.000
					3	431	304	62	0.084
					4	2017	1307	7867	2.367

* - less than 1 second

Table 2: Generation Time of All Maximal Low Degree Patterns

1. It can be seen that there is a substantial difference between the times reported in Tables 1 and 2. On the one hand Table 2 reports the results of enumerating two families of maximal patterns, the positive and the negative ones, while Table 1 is only concerned with one family. On the other hand, and perhaps more importantly, Table 1 reports only the time required for enumerating all intervals and calculating their prevalences, while Table 2 includes the evaluation of the patterns for feasibility, the comparisons of all pairs of patterns for the detection of possible inclusion relations, and their actual listing.

2. Table 2 clearly shows the efficiency of the proposed method. It can be seen that the enumeration of all maximal patterns of degree 2 takes always less than 1 second. Similarly, the generation of all maximal patterns of degree 3 takes in most of the cases less than 2 minutes, and even in the worst case (**hea**) less than 4 minutes. Finally, the generation of all maximal patterns of degree 4 takes in most of the cases less than 1 hour, and only one of the experiments needed more time.

The average time requirements for the generation of a maximal pattern are also very instructive. It follows from Table 2 that the average time needed for the generation of a maximal pattern of degree 3 is of 0.128 seconds, and that that needed for the generation of a maximal pattern of degree 4 is of 3.26 seconds.

The most important conclusion following from all the above is that the enumeration of all maximal patterns can be accomplished in a very short time. This can have important consequences not only on the efficiency of LAD, but on that of various other data mining procedures.

3. Finally, the experiments show clearly the "explanatory power" of the family of all small degree patterns. A good measure of the explanatory power of a family of patterns is the number of those points in the training and in the validation sets which belong to at least one of the patterns in that family.

It can be seen in Table 3 that the family of all patterns of degree 2 covers on the average more than 86% of the points in the observation sets (excepting the notoriously difficult **bld** problem, but including **pid** which is also well-known to be very difficult). Moving to the family of patterns of degree 3 we can see that they cover on the average more than 95% of the points in the datasets. Finally, the family of patterns of degree 4 covers more than 98% of the datasets.

Degree	Dataset	Number of Positive Observations			Number of Negative Observations			Number of All Observations		
		Training	Validation	Total	Training	Validation	Total	Training	Validation	Total
2	**bcw**	100%	100%	100%	97%	98%	98%	98%	99%	98%
	vot	96%	95%	96%	**	**	**	**	**	**
	bld	78%	73%	76%	30%	18%	24%	58%	50%	54%
	hea	100%	99%	99%	100%	99%	99%	100%	99%	99%
	pid	85%	83%	84%	99%	97%	98%	95%	93%	94%
3	**bcw**	100%	100%	100%	99%	99%	99%	99%	99%	99%
	vot	99%	98%	99%	98%	98%	98%	99%	98%	98%
	bld	93%	89%	91%	77%	57%	67%	86%	76%	81%
	hea	100%	99%	99%	100%	100%	100%	100%	99%	100%
	pid	93%	93%	93%	100%	99%	100%	98%	97%	98%
4	**bcw**	100%	100%	100%	100%	100%	100%	100%	100%	100%
	vot	100%	98%	99%	100%	100%	100%	100%	99%	100%
	bld	96%	95%	96%	89%	86%	88%	93%	91%	92%
	hea	100%	100%	100%	100%	100%	100%	100%	100%	100%
	pid	97%	99%	98%	100%	99%	100%	99%	99%	99%

** - No negative feasible patterns of degree 2 exist for the **vot** dataset.

Table 3. Coverage of the Observations by Families of Maximal Patterns

As a final conclusion we mention the fact that the family of all low degree patterns can be generated in a very reasonable time, and covers all the observation points in the 5 datasets examined.

6 References

[1] J. Boothroyd, Algorithm 246, Gray code. *Comm. ACM 7*, 12 (1964), 701.

[2] E. Boros, P.L. Hammer, T. Ibaraki, A. Kogan, Logical Analysis of Numerical Data, *Mathematical Programming*, 79, 1997, pp. 163-190

[3] E. Boros, P.L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, I. Muchnik, An implementation of Logical Analysis of Data, *IEEE Transactions on Knowledge and Data Engineering*, 12, No. 2, 2000, 292-306

[4] G. Ehrlich, Loopless algorithms for generating permutations, combinations and other configurations. *J. ACM* 20, 3 (1973), 500-513.

[5] Y. Crama, P.L. Hammer, T. Ibaraki, Cause-effect relationships and partially defined Boolean functions, *Annals of Operations Research*, 16 (1988) 299-326
[6] Flores, Reflected number systems. *IRE Trans. Electron. Comput.* 5, 2 (1956), 79-82.
[7] E.N. Gilbert, Gray codes and paths on the *n*-cube. *Bell Syst. Tech. J.* 37, 9 (1958), 815-826.
[8] T.S. Lim, W.Y. Loh,Y.S Shin, A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms, *Machine Learning*, 40, 203-229 (2000)
[9] J.W. Lenstra, A.H.G. Kan Rinnooy, *A recursive approach to the generation of combinatorial configurations*. Rep. BW 50.75, Mathematisch Centrum, Amsterdam, 1975.
[10] P.L. Hammer, Partially defined Boolean functions and cause-effect relationships, *International Conference on Multi-Attribute Decision Making Via OR-Based Expert Systems*, University of Passau, Passau, Germany, 1986.
[11] Y. Koda, F. Ruskey, A Gray code for the ideals of a forest poset, *Journal of Algorithms* 15 (1993) 324-340.
[12] J. Misra, Remark on Algorithm 246. *ACM Trans. Math. Software* 1, 3 (1975), 285.